This blank page was inserted to preserve pagination.

MAC-TR-4

MASSACHUSETTS INSTITUTE
OF TECHNOLOGY

PROJECT MAC

VERBAL AND GRAPHICAL LANGUAGE FOR THE AED SYSTEM: A PROGRESS REPORT

by

DOUGLAS T. ROSS and CLARENCE G. FELDMANN

This empty page was substituted for a blank page in the original document.

This empty page was substituted for a blank page in the original document.

This empty page was substituted for a blank page in the original document.

VERBAL AND GRAPHICAL LANGUAGE FOR THE AED SYSTEM: A PROGRESS REPORT

INTRODUCTION

Since 1959 the Computer Applications Group of the Electronic Systems Laboratory and members of the Design Division of the Mechanical Engineering Department have been working on the Computer-Aided Design Project sponsored by the U. S. Air Force. The objective is to create a man-machine system in which a group of designers and a computer can work together as a team on fresh design problems which require creative solutions. Since the concepts of time-sharing and dynamic man-machine interaction are inherent in the concept of Computer-Aided Design, this work is also being supported by Project MAC as an integral part of its general goals as well.

The Computer-Aided Design System is not intended to solve any particular class of problems, but instead should be applicable to essentially any area of design and problem-solving. In order to achieve this generality with a single comprehensive system requires careful attention to the fundamentals of solving problems with computers. Since the system is to be applicable to essentially any kind of problem, a single unified approach to handling the data and information about problems is required. Thus one of the early developments of the Project was the concept of a technique for not only containing all of the data about a problem, but also showing all of the requisite interrelationships among the individual items of data in what is called a plex structure.

"Plex" is derived from the word "plexus" which has a dictionary meaning "an interwoven combination of parts in a structure; a network". A plex is considered to be composed of elements of various types, each type of element having a number of components appropriate to the object or relationship which the element represents. Components of elements may contain data in numerical or coded form, or pointers to other elements. In application, the objective is to model all of the pertinent information about a problem in an elaborate plex structure so that all of the data and relationships are explicitly shown. If this can be accomplished,

then any processing algorithm can obtain any information it requires by suitable referencing of the elements in the plex structure. Figure 1 shows the modelling plex for a line in two-dimensional cartesian co-ordinate space. The elements contain type and name components, as well as pointers to show the end-point relationships, and places for storing the coordinate values. Any property of the line which is required, such as its length or its slope, may be computed by referencing the appropriate components in the elements of the modelling plex.

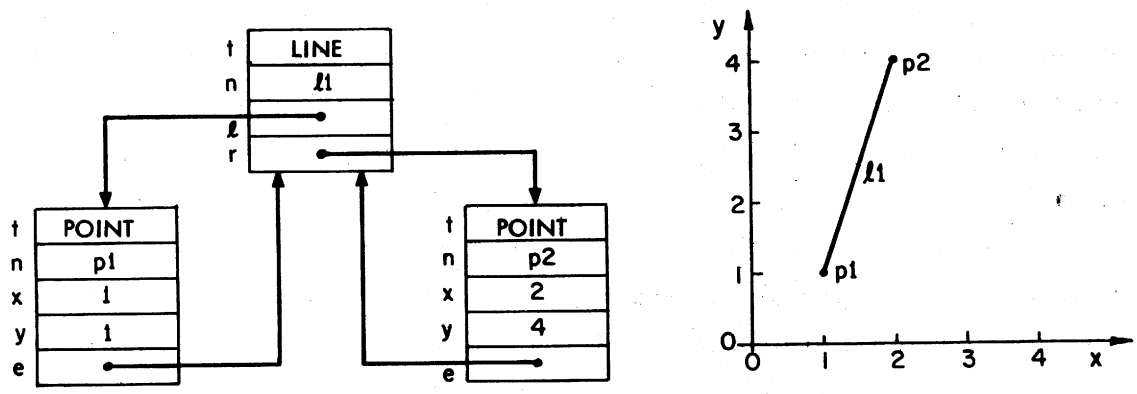


Fig. 1 Modelling Plex for 2-D Line

The concept of plex structures is well suited to the requirements for providing a firm foundation for development of the Computer-Aided Design System, since it is general, powerful, and may be mechanized in a great many ways on computers, but for any actual problem, the plex structures which arise are so elaborate and complex as to be essentially incomprehensible to humans. Thus a mechanism is needed for automatically transforming the ideas which a human designer may have about problems into the intricacies of the modelling plex.

The concept of plex actually involves more than mere structure or form. There is no unique modelling plex for an object or a piece of problem in general. Instead the structuring and choice of components is determined by the use which is to be made of the model. In other words an inherent part of the concept of plex is the idea that processing algorithms will interpret the contents of the components and thereby ascribe meaning or purpose to them. What the components represent depends very closely upon the algorithms which reference them.

Of great importance are the algorithms which describe the process whereby individual elements are assembled to form a complicated plex. The vast complexity of a modelling plex never arises all at once, but instead is built up step-by-step by a process of accretion. The viewpoint is that special meta-properties are ascribed to the elements and these meta-properties control the behavior of algorithms which establish the step-by-step interconnection of elements to cause the growth of a large structure. The effect of individual elements being assembled by an algorithm into a large structure is as though the combination of the meta-properties and the algorithm gave behavioral properties to the elements themselves, so that the elements interact to form large structures in much the same way that chemical elements interact to form large molecules.

In this short progress report we try to demonstrate how this abstract concept of plex as a mixture of structure and behavior can be applied to yield efficient and powerful mechanisms for solving the numerous problems involved in research on the Computer-Aided Design System. The technique has been applied within the Project in a great many places, but here we consider only the problems of verbal and graphical language and the compilation of efficient computer programs.

LANGUAGE

It was mentioned above that in order to make use of the plex concept within the Computer-Aided Design System it was necessary to achieve a mechanism for going automatically from the way in which it is natural for a human to think of a problem into the modelling plex form. What is desired is a language which will seem very natural to the human and yet which can automatically be processed so that the meaning of the statements made in the language can efficiently be transformed into the modelling plex.

There is a vast difference between a statement about something and the something itself. This truism shows clearly in the over-all scheme for the Computer-Aided Design System. The idea that meta-properties may be ascribed to objects which then will control the behavior of algorithms to build large structures out of those objects has been applied to the problem of language, and has led to what we call the Algorithmic Theory of Language. Given a vocabulary consisting of any number of words, meta-properties are assigned to each word, such that

the "First-Pass Algorithm" will automatically construct, for any grammatical statement using those words, a modelling plex for the statement. This plex explicitly exhibits the syntactic and semantic structure of the statement and is called the first-pass structure for the statement. It shows the right and left context for each vocabulary word in the statement, in the form of a parsed tree structure, and in addition by means of a chain of pointers called the precedence string, shows the precise order in which the words should be considered in order to build up the meaning of the entire statement from the meaning of its subparts, step-by-step. As is shown in Figure 2 the meaning of the statement is then transformed into changes in the modelling plex proper by means of "operators" which follow the precedence string and transform the information contained in the first-pass structure into changes in the modelling plex. Thus the first-pass structure models the statement about something while the modelling plex models the something itself.

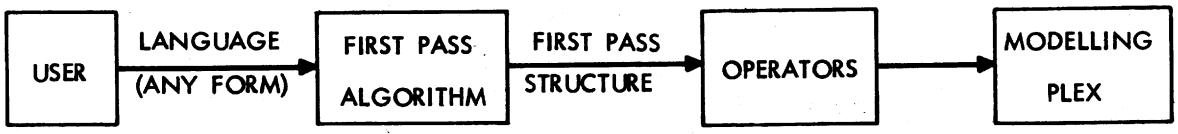


Fig. 2 The Role of Language and Operators

The language theory has been described elsewhere (see references) and is still in a state of flux and further development, and will not be considered further in detail here. Instead the remaining portions of this paper are concerned with applications of the theory in the over-all Computer-Aided Design framework.

THE AED-0 COMPILER

It was found that no existing compiler could provide an efficient mechanization of the plex concept so that in the Spring of 1963 the Project undertook to construct a compiler incorporating the techniques proposed for the Computer-Aided Design System itself as the first major step toward accomplishing the goals of the Project. Prior to that time plex programming had been carried out using an experimental compiler called the Bootstrap Compiler, which also was written by the Project, both for educational experience and as an experimental tool. The AED-0 Compiler

was originally written in the Bootstrap Compiler language, but recently the resulting system has been disassembled from binary machine code back into FAP assembly language and a great many changes and improvements to the AED-0 Compiler have been made so that now it is available as a public command in the Project MAC Time-Sharing System.

AED-0 language at present consists of Algol-60, with some features omitted, and othersfor plex programming added. The compiler is very efficient and flexible and compiles very good machine code in most cases. The compiler is based directly on the plex concepts described above. The following illustration pictures the AED-0 compilation process:

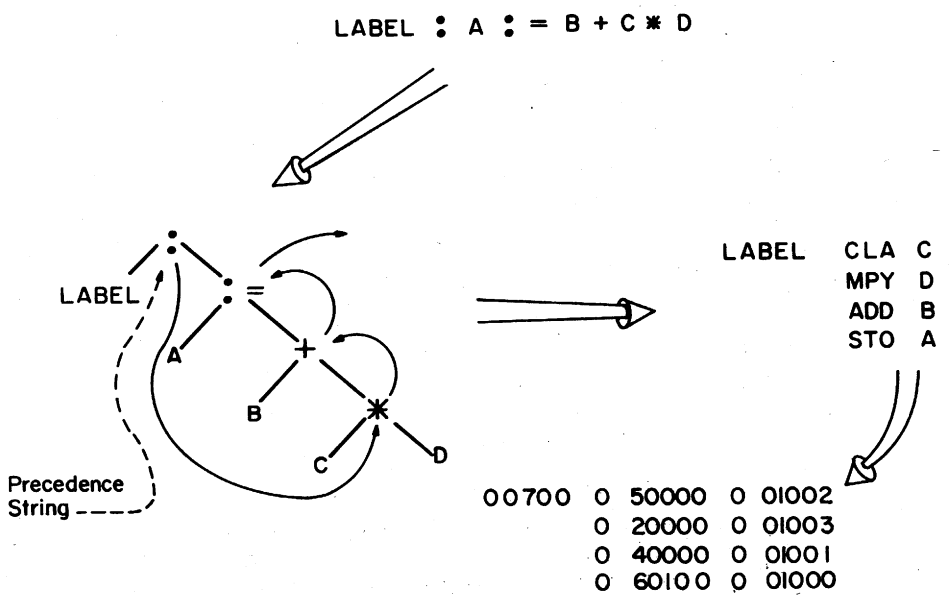


Fig. 3 AED-O Data Flow

Programmer statements are read by AED-0, and transformed into the first-pass structure which represents the machine's understanding of the statements. Note in the illustration the tree structure which gives the syntactic structure of the words in the input string, and the "precedence string" which threads through the tree structure, indicating the semantic structure of the words by specifying the exact order in which the words must be considered in order for the proper meaning to be obtained. Operators may be written to follow the precedence string and perform desired operations.

One such operator is a machine-language compilation operator which is used by AED-0 to produce a binary object program which may be executed by the computer. Because of the convenient plex form of the statement, this operator is extremely efficient, and in a single pass, produces the object code from the precedence string.

The method used by this compilation operator follows the AED philosophy that large, complex problems do not arise all at once, but are built up step-by-step out of small, simpler pieces. The basic building blocks are the individual vocabulary words, and in AED-0 these comprise the (approximately) 100-word Algol-60 vocabulary plus additional words needed to define and manipulate plex structures conveniently.

The function of the compilation operator is therefore to follow the precedence string, and as each node in the tree structure is reached, to merge an additional segment with the object deck. These segments are known as "merbes" (merge beads) since they are thought of as beads on a string which are designed to be merged with other such beads.

The individual vocabulary merbes and the object deck are of identical construction, so that the compilation process is to merge small merbes into a single large merbe of the same form. Therefore, at any time during the compilation the object deck may be packaged, given a name in the vocabulary table, and merged with other compilation merbes. Thus AED-0 has no "object code generator" as such, but instead constructs the object program by the uniform process of "merging".

The following is a diagram of a merbe. The merbe illustrated does not correspond to any Algol vocabulary word, but illustrates several features of the merbe structure.

Note that the merbe is divided into a "head" which contains the required information about the merbe's dummy arguments, and a "tail" which contains the actual machine code operations corresponding to the related vocabulary word. Corresponding to the vocabulary words, AED-0 contains merbes which, if merged with the object merbe, will perform the desired operations when executed by the computer. At merging time, the specific variable names used by the programmer are "substituted" for the merbe's dummy arguments by linking together the chains of pointers shown.

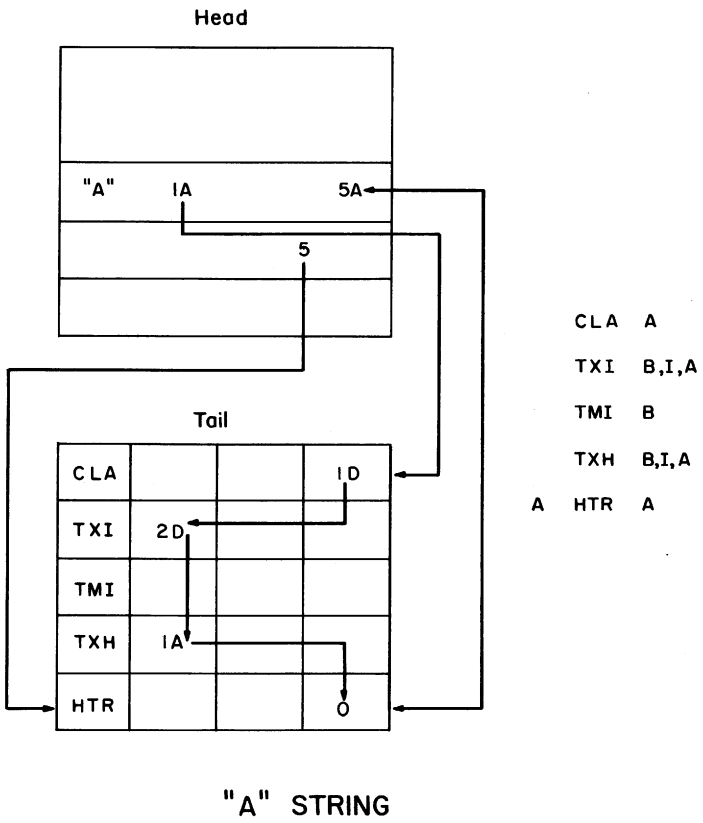


Fig. 4 MERBE Structure

In the illustration, the dummy argument is labelled "A". We see that this dummy is used in the four places illustrated in the merbe tail, and is assigned to the fifth tail location. The head component for the dummy A tells the first and last usage of A in the tail, and the assigned location (if any) of A, relative to the start of the tail. The tail contains the usage string showing, at each position, where the next usage of the symbol occurs. This usage string is divided into the computer's machine-code word divisions, as illustrated. Each such usage pointer is relative to its own location, and contains a code telling the subdivision of the computer word in which to find the next usage.

The process of merging a vocabulary merbe with the object code merbe is illustrated on the next page.

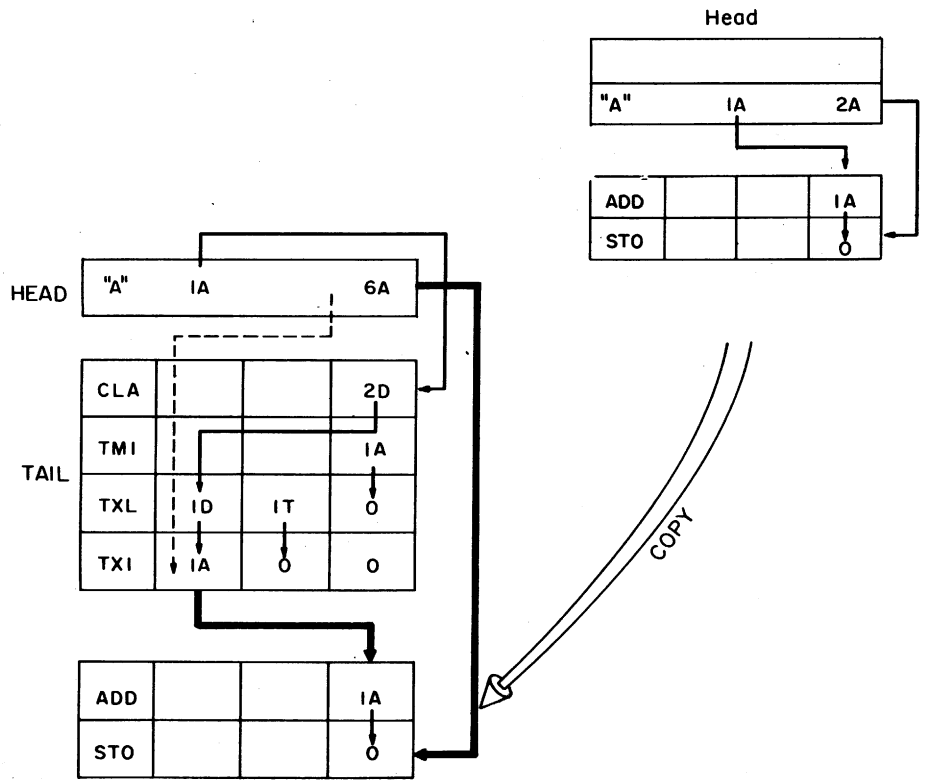


Fig. 5 MERGE Process

However large the merbe, the operation is the same. A copy of the tail of the vocabulary merbe is appended to the object merbe. Then for each dummy argument the operations required (heavy black arrows) are to reset two pointers, the last-usage pointer and the final pointer in the tail string for the symbol used. The previous last-usage pointer is illustrated by the dotted arrow.

When the entire compilation is completed, the object merbe is in a convenient form for allocating memory locations for all variables and constants used in any way. Once values are selected, it is only necessary to follow the tail strings, plugging in the assigned values at each point on the string.

The allocatable (merbe) form of object program is more general than the standard BSS format used with current IBM machines, and is more easily adapted to splitting binary object decks and re-distributing

program pieces in a time-shared core memory environment. However, in order to be compatible with other existing compilers at Project MAC, AED-0 transforms the object merbe into the standard BSS binary format for output.

In summary, the AED-0 Compiler applies the basic plex concepts in many places and in many forms. In particular the first-pass of the compiler is the First-Pass Algorithm of the language theory, and the final merging operation of the compiler is a direct application of step-by-step construction of an appropriate mechanization of a modelling plex by operators following the precedence string.

EXAMPLE OF AN AED-0 PROGRAM USING CTSS

Introduction

The following is an example of an AED-0 program. The program illustrated does not attempt to show the full power of the AED-0 language or forecast any of the advanced features planned for AED-1. It is an attempt to show that AED-0 is a functioning compiler which may be used now as a tool in CTSS.

Input Disk File

The following is the printout of a CTSS input file of a subroutine to calculate and print out the solutions to a quadratic equation of the form $Ax^2 + Bx + C = 0$. Tests are made for $A = 0$, $B^2 - 4AC$ negative, $B = 0$, and $B^2 - 4AC = 0$. In each case, the proper alternate solution and corresponding printout are performed.

```
printf quadr algol
W 1425.1
00010 BEGIN
00020 DEFINE PROCEDURE QUADRATIC(A,B,C) WHERE
00030 REAL A,B,C TOBE BEGIN
00040 REAL X,TEMP $,
00050 IF A EQL 0 THEN BEGIN X=-C/B $, PRINT ONEVAL,X
00060 $, GOTO RETURN END $,
00070 IF B*B-4.0 * A*C LES 0 THEN BEGIN
00080 PRINT NEGATIVE $, GOTO RETURN END $,
00090 LISTING $,
00100 X=(-B+(TEMP=(SQRT(TEMP=B*B-4.0 *A*C))))/(2.0 * A) $,
00110 LISTING $,
00120 IF B EQL 0 THEN BEGIN X= ABS X $, PRINT ONLY,X END
00130 ELSE IF TEMP EQL 0 THEN PRINT ONEVAL,X ELSE BEGIN
00140 TEMP=(-B-TEMP)/(2.0 * A) $, PRINT BOTH,X,TEMP $,
00150 END $, END $,
00160 ONLY $ FORMAT(20H SOLUTION X= + OR - ,F5.2) $,
00170 BOTH $ FORMAT(13H SOLUTION X= ,F5.2,2X,5H AND ,F5.2) $,
00180 NEGATIVE $ FORMAT(13H NO SOLUTION.) $,
00190 ONEVAL $ FORMAT(20H SINGLE SOLUTION X= ,F5.2) $,
00200 END FINI
R .600+.800
```

Compilation and PRALG Listing

The following illustrates the compilation of the QUADR file, using AED. The commands (PRLG) and (SYMB) instruct AED-0 to create printed output files for the Print Algorithm version of the input data and resulting symbol table, respectively.

The later printout of the PRALG file is also illustrated. Note that the PRALG program has re-arranged the page layout to correspond to the BEGIN-END block structuring of Algol, and has placed statement labels at the left margin for easy reference. This print is produced by an operator which prints directly from the tree structured machine version of the input statements, and thus mirrors the machine understanding of what was said, not merely a re-play of the input file. Its readable format is of great assistance in making clear the structure of the program.

```
aed quadr (prlg) (symb)
W 1432.9
 LENGTH = 00247 ENTRY = 00003
R 2.516+2.000

ctest8 quadr pralg 14
W 1435.6
1PAGE 1


              BEGIN
              DEFINE PROCEDURE QUADRATIC ( A , B , C ) WHERE REAL A , B , C TOBE
                   BEGIN
                   REAL X , TEMP $,
                   IF A EQL 0 THEN
                        BEGIN
                        X = - C / B $,
                        PRINT ONEVAL , X $,
                        GOTO RETURN
                        END $,
                   IF B * B - 203400000000 * A * C LES 0 THEN
                        BEGIN
                        PRINT NEGATIVE $,
                        GOTO RETURN
                        END $,
                   LISTING $,
                   X = ( - B + ( TEMP = ( SQRT ( TEMP = B * B - 203400000000 * A
                   * C ) ) ) / ( 202400000000 * A ) $,
                   LISTING $,
                   IF B EQL 0 THEN
                        BEGIN
                        X = ABS X $,
                        PRINT ONLY , X
                        END
                   ELSE IF TEMP EQL 0 THEN PRINT ONEVAL , X ELSE
                        BEGIN
                        TEMP = ( - B - TEMP ) / ( 202400000000 * A ) $,
                        PRINT BOTH , X , TEMP $,
                        END $,
                   END $,
 ONLY $       FORMAT(20H SOLUTION X= + OR - ,F5.2)        $,
 BOTH $       FORMAT(13H SOLUTION X= ,F5.2,2X,5H AND ,F5.2)    $,
 NEGATIVE $   FORMAT(13H NO SOLUTION.)        $,
 ONEVAL $     FORMAT(20H SINGLE SOLUTION X= ,F5.2)        $,
              END FINI
R 1.200+1.616
```

Symbol Print

The following is a printout of the QUADR file symbol table, generated by the (SYMB) command above. This printout is primarily useful in machine language debugging, when necessary.

```
ctest8 quadr symbol 14
W 1439.4
1PAGE 1      SYMBOL TABLE


 SYMBOL TABLE NAME = REALS
202400000000 = 00237      203400000000 = 00240

 SYMBOL TABLE NAME = INTS
0 = 00241

 SYMBOL TABLE NAME = TEMPS
(L007 = 00161  (L006 = 00140  (L005 = 00161  (L004 = 00123  (L003 = 00053
(C007 = 00242  (C006 = 00243  (L002 = 00025  (L001 = 00004  (L000 = 00163
(C005 = 00244

 SYMBOL TABLE NAME = SYSTEM
SQRT = 00000                              (FIL) = 00001
(SPH) = 00002

 SYMBOL TABLE NAME = ST001
BOTH = 00222                              ONLY = 00231
ONEVAL = 00210                            NEGATIVE = 00216
QUADRATIC = 00003

 SYMBOL TABLE NAME = QUADRATIC
A = (ARG)                                 B = (ARG)
C = (ARG)                                 X = 00245
TEMP = 00246

LENGTH = 00247 ENTRY = 00003

TRANSFER VECTOR
SQRT = 00000         (FIL) = 00001        (SPH) = 00002

ENTRY POINTS
QUADRA = 00003
R 1.000+.800
```

Compilation Listing

The following is a compilation listing of the object code produced by the single statement in line 100 of the input QUADR file. The statement LISTING $, in lines 90 and 110 caused the listing of line 100 to be produced (see input file print, above).

```
ctest8 quadr list 14
W 1441.8
1PAGE 1       COMPILATION LISTING


 00053                                  CLA    B
 00054                                  CHS    2
 00055                                  STO    (C006
 00056                                  LDQ    B
 00057                                  FMP    B
 00060                                  STO    (C007
 00061                                  LDQ    203400000000
 00062                                  FMP    A
 00063                                  XCA
 00064                                  FMP    C
 00065                                  STO    (C005
 00066                                  CLA    (C007
 00067                                  FSB    (C005
 00070                                  STO    TEMP
 00071                                  TSX    SQRT,4
 00072                                  STO    TEMP
 00073                                  FAD    (C006
 00074                                  STO    (C006
 00075                                  LDQ    202400000000
 00076                                  FMP    A
 00077                                  STO    (C005
 00100                                  CLA    (C006
 00101                                  FDP    (C005
 00102                                  STQ    X
R 1.000+2.200
```

Object Deck Execution

The following illustrates the execution of the QUADR subroutine, along with a main program which feeds values of A, B, and C from the teleprinter. Each of the four special cases tested for in the input deck are illustrated.

```
loadgo main quadr
W 1426.8
EXECUTION.
                QUADRATIC SOLUTION PROGRAM


 PLEASE TYPE VALUES OF A,B, AND C
 A =
1.
 B =
1.
 C =
1.
 NO SOLUTION.


 PLEASE TYPE VALUES OF A,B, AND C
 A =
1.2
 B =
3.9
 C =
-7.4
 SOLUTION X=  1.34   AND -4.59


 PLEASE TYPE VALUES OF A,B, AND C
 A =
2.0
 B =

 C =
-2.0
 SOLUTION X= + OR -  1.00


 PLEASE TYPE VALUES OF A,B, AND C
 A =
0.0
 B =
7.9
 C =
7.9
 SINGLE SOLUTION X= -1.00
```

GRAPHICAL LANGUAGE

It is of vital importance that the language facility for the Computer-Aided Design System include not only flexible descriptive and programming languages in word form, but a generalized capability for graphical communication as well. There are many aspects of design in almost any field, for which the natural means of expression is in terms of pictures or diagrams, and any attempt to convey equivalent information in verbal form would be extremely unnatural and awkward, and would defeat the basic principle that the designer-user be able to operate in a manner which is natural to him.

The ESL Computer Applications Group has been active in the field of on-line man-machine systems for over 10 years (the first tracking program was written in late 1954 for the Whirlwind Computer), but the first complete subsystem for graphical communication was the Sketchpad program of Dr. I. E. Sutherland, written for the TX-2 Computer at Lincoln Laboratory, with NSF and Lincoln Laboratory support, in 1962. This program is one of the outstanding success stories in the field of computer applications, for it made the concept of graphical communication with a machine come alive in a very meaningful way to many thousands of people.

Sketchpad and the plex concept which underlies the AED System share a common heritage. In late 1961 Sutherland had completed his first attempt at a light-pen drafting language, based upon the use of tables of points and lines, and push-button commands corresponding to standard drafting tools for drawing horizontal and vertical lines, slanted lines, circles, etc. At the same time the Project was independently beginning to apply the concepts of the Bootstrap Compiler to the consideration of graphical language and was beginning a study of a "Bootstrap picture language." At that time, the plex concept was thought of as almost purely structural, and although a preliminary version of the First-Pass Algorithm for programming languages of the Algol type had been devised in the preceding months, the strong relationship between interaction algorithms and structure was not then apparent. Still earlier, in 1960, the Project had carried out a "point-line diagram study" in order to gain experience with list processing techniques. This problem concerned techniques for constructing diagrams composed of points, lines, and angles, and imposing geometric constraints on the elements of the diagram

in successive stages, as an example both of graphical language and as a model for the design process itself. The problem was carried out in the LISP system then being constructed by the MIT Artificial Intelligence Group and no attempt was made to drive the resulting programs with light-pen inputs. Instead the study provided impetus to the developments of the more general plex concepts, since it was felt that the storage and time expenditures inherent in attempting to model things entirely in terms of lists and trees would be impractical for a commercially feasible Computer-Aided Design System.

In early 1962, then, the interaction between the beginning First-Pass Algorithm and the Bootstrap Picture Language led the Project to the generalizations which evolved into the Algorithmic Theory of Language itself, while Sutherland, influenced by the structural aspects of the plex concept of that time, pursued Sketchpad proper. The Bootstrap Picture Language study as such was discontinued in view of the success of Sutherland's efforts.

With the successful on-line operation of the ESL Display Console on the Project MAC Computer, attention has now returned to the problem of providing graphical language capabilities on commercial equipment as a part of the Computer-Aided Design System. As a beginning the highly successful Sketchpad capability will be duplicated externally. The internal processing of the programs, however, will be almost entirely different from those used by Sutherland. Whereas Sketchpad was created on the TX-2 Computer through the considerable programming artistry of Sutherland using the TX-2 macro assembly system, graphical language for the AED System will be mechanized as a special application of the Algorithmic Theory of Language and plex concepts. Therefore, the original objective that there should be no distinction between verbal and pictorial language for the Computer-Aided Design System will be achieved.

In order not to interfere with the compiler developments of the Project, and in order to obtain a simpler base of programs to work from, while at the same time providing an experiment in dynamic man-machine interaction in a time-sharing environment, a simplified miniaturized version of the over-all Computer-Aided Design System has been written. Although not yet officially christened, this little system will be referred

to as AED Jr. in the following description of how it has been used to prepare a preliminary demonstration of Sketchpad capabilities within the over-all AED framework.

AED Jr. consists of a master control program and a number of sub-programs for setting up the meta-properties of new vocabulary words, examining the vocabulary table entries, making corrections, running statements through the First-Pass Algorithm, and examining the syntactic structure in the form of the parsed tree, and checking the correctness of the precedence string which models the semantic structure of statements. All of these features are directly under the control of a simple command language which may be typed on the teletype, and included among these commands are commands to accept input statements from the light pen and push buttons of the ESL Display Console and to plot graphical statements on the console. In the following description characters printed by the system are in upper case and characters typed by the user are in lower case. We describe the features of the system by illustrating how a trivial language consisting of the words begin, end, and fini may be inserted and tested, and then give some illustrations of the results of the more elaborate graphical language used for the May 6th demonstration.

AED Jr. is called in for execution just as any other program in the time-sharing system. After a few preliminaries the program types out

```
MASTER
TYPE.
```

This indicates that AED Jr. is now at the master control level and is awaiting instructions. The word TYPE. is not followed by a carriage return so that any typing which we do will appear on the same line and will be commands to the system. If we type "vio", calling for the vocabulary in-out subsystem, AED Jr. responds

```
MASTER
TYPE. vio
WHAT
TYPE.
```

The word what indicates that we are now in a master control for the vio routine and may perform any of its functions. In general when the system is operated on-line from the teletype, words such as master and what are printed out by AED Jr. to inform the user of the status of the system, and to prompt him as to what he may do next. In order to save console response time, any number of user words may be typed on a single line, separated by spaces, in which case the system words are typed out down the page as actions are taken. The system also may be operated off-line from user commands prepared through the CTSS Input and Edit facility, in which case the system comment words such as what do not appear.

To check that the system contains no preliminary vocabulary let us ask it to show us all of the initial vocabulary table:

```
WHAT
TYPE. show all
WORD
NIL     COD    0 LT 000 RT 000 VR 000 GN 000
WHAT
TYPE.
```

AED Jr. shows us that the vocabulary at present consists of only the empty type nil with zeroes uniformly for all of its meta-properties. Continuing with our check that the system is starting with nothing we add some more commands.

```
WHAT
TYPE. out state
MASTER
N IS NIL
X IS NIL    ...T OF X IS NIL
P EMPTY
MASTER
TYPE.
```

which shows that we have gone out of the vio subsystem back to the master system and have asked for the state of the First-Pass Algorithm. AED Jr. reports to us that the state variables of the First-Pass Algorithm, n, x, and p, all contain nil and at the stack of uncompleted things is empty, and furthermore, that the type of the thing in x (nil) is nil. Thus we start with a clean slate.

To insert our trivial three-word vocabulary we return to the vio routine and insert our three words,

```
MASTER
TYPE. vio vin x 000001 nil nil nil nil nil
WHAT
VIN
VIN
TYPE. begin 000002 x x x x x
VIN
TYPE. end 000003 x x x x x
VIN
TYPE. fini 000004 x x x x x out
VIN
WHAT
TYPE.
```

The word vin in the vio system says that we wish to put in a vocabulary word. Note that we have defined a dummy word x to save typing as we provide the meta-properties for the words of our trivial vocabulary. With the words now entered into the vocabulary table we need to complete the specification of the meta-properties by specifying for each vocabulary word what kinds of things does it like to have on its left and right sides, i.e., in what kinds of contexts can these words be used. We continue

```
WHAT
TYPE. like begin nil right end out
WORD
LEFT SET
WORD
TYPE.
```

Note that vio responds to the command like by asking us what word to refer to. When we reply begin, it informs us that it is set to establish what that word likes on its left. We specify nil and then say we would like to set the right likes and that on the right begin is to like the word end. Then we wish to go out from the setting of likes for that word. The like routine responds by asking us what is the next word that we would like to treat? We continue

```
WORD
TYPE. end nil right nil out fini begin out out
LEFT SET
WORD
LEFT SET
WORD
WHAT
TYPE.
```

Note that fini likes the word begin on its left, but we specify no right likes for fini. Now we may check that our vocabulary is properly set.

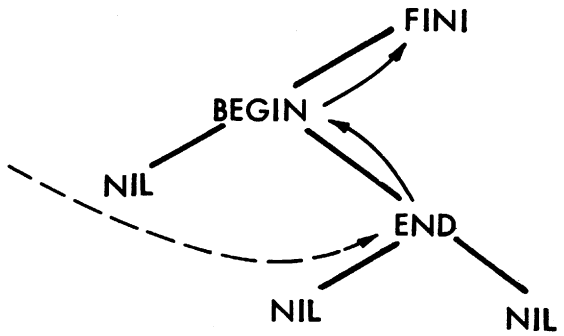
```
WHAT
TYPE. show all out
WORD
X       COD    1 LT NIL RT NIL VR NIL GN NIL
NIL     COD    0 LT 000 RT 000 VR 000 GN 000
BEGIN   COD    2 LT X   RT X   VR X   GN X
  LEFT  NIL
  RIGHT END
END     COD    3 LT X   RT X   VR X   GN X
  LEFT  NIL
  RIGHT NIL
FINI    COD    4 LT X   RT X   VR X   GN X
  LEFT  BEGIN
WHAT
MASTER
TYPE.
```

Actually there is only one grammatical statement which can be made out of our little vocabulary so we try it:

```
MASTER
TYPE. fresh run begin end fini state sim
FRESH SET
MASTER
MASTER
N IS FINI
X IS          ...T OF X IS X
BEGIN  END      NIL
NIL    NIL

P EMPTY
MASTER
  1
END      3
BEGIN    2
MASTER
TYPE.
```

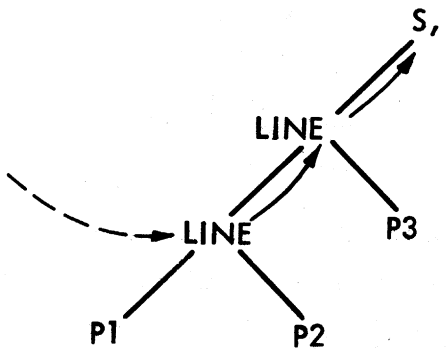
which says that we wish to have a fresh state of the algorithm, to try a run consisting of the statement "begin end fini" and we wish to see the state of the First-Pass Algorithm and simulate a run along the precedence string in the first-pass structure. For each word there is a possibility of a "left execute" using only the left context and a possibility of a "right execute" action using the right context. In the simulation the code 1 means "do the left only", 2 means "do the right only", and 3 means "do both left and right". The teletype information provided by AED Jr. is equivalent to the following first-pass structure.



If we had encountered any difficulties we could recall the vio routine, change the meta-properties or the vocabulary words themselves, return to master and request a "fresh run". In this way without ever leaving the system we can establish and immediately check out essentially any programming language of our choosing.

To apply AED Jr. to the problem of writing graphical language requires some additional machinery. The ESL Display Console is connected directly to the direct data channel of the MAC 7094 Computer and a real time program to control the console, called the A-core Program is incorporated into the time-sharing supervisor. The A-core Program in turn communicates with the "B-core Program" which shuttles in and out in time-sharing with the user's program (in this case AED Jr., etc.). It is necessary to replace the run subroutine by a draw subroutine so that the input string statement may be obtained from button-pushings and light-pen motions in the display console via the A-core-B-core route. Similarly, the sim function is replaced by a subroutine called plot which replaces the printing function of sim by using the left and right executes for the vocabulary words to transform the words encountered in the first-pass structure into the appropriate display commands for transmittal to the display console via the B-core-A-core route.

In the case of graphical language, whenever a button at the console is pushed, it stands for a word, such as point or line, and the draw routine is set so that when a button is pushed, not only is the associated word transmitted to the First-Pass Algorithm of AED Jr., but also the appropriate light-pen information, as indicated by the VR component of the meta-properties of the word. Thus if the button standing for line is pushed, the draw routine will send along to the First-Pass Algorithm the current light-pen location as an atomic variable, followed by the word line. If the same button is pushed again after the pen has been moved, the new pen location and new word line are passed along. Finally pushing a terminate ($,) button sends along a final pen location with the terminate word so that the following first-pass structure results.



The plot function can then make up display commands for a line from p1 to p2 and another line from p2 to p3.

Although only a very limited vocabulary is incorporated into the May 6th Demonstration Program, some of the available functions are indicated in the accompanying illustrations. Many additions to the system will be made in the next few months, including facilities for generalized constraint satisfaction.

It is interesting to note that several features, such as the fact that the topology of a picture is maintained as the atomic points are moved around, which were handled by the generalized constraint satisfaction facility in Sketchpad are here only an aspect of the phrase-structuring of the graphical language. The use of subpictures as elements of bigger pictures, which was the principle feature of the Bootstrap Picture Language and which also appears in Sketchpad, is closely related to the ideas of subroutines which can be called with variable arguments in compiler language. Many similar relationships can be seen between the verbal

```
MASTER
TYPE. run begin aa line bb arc cc point dd to ee line
TYPE. ff $, gg move hh point ii $,
TYPE. jj erase kk arc ll to mm arc nn to oo $,
TYPE. pp end fini state sim
MASTER
N IS FINI
X IS        ...T OF X IS X
BEGIN  END     NIL
NIL    |
       |
       $,      PP
       |
       $,      ARC     TO      OO
       |       |       NN
       |       |
       |       ARC     TO      MM
       |       |       LL
       |       |
       |       ERASE   KK
       |       JJ
       |
       $,      POINT   II
       |       |
       |       MOVE    HH
       |       GG
       |
       LINE    FF
       |
       ARC     TO      EE
       |       |
       |       POINT   DD
       |       CC
       |
       LINE    BB
       AA
```

Fig. 6 Example of Parsed Tree of Graphical Language in AED Jr.

```
MASTER
TYPE. fresh run begin aa arc bb point cc line dd arc
FRESH SET
MASTER
TYPE. ee point ff arc gg to hh point ii to jj point
TYPE. kk to ll $, mm end fini state sim
MASTER
N IS FINI
X IS        ...T OF X IS X
BEGIN  END     NIL
NIL    |
       |
       $,      MM
       |
       ARC     TO      LL
       AA      |
               |
               POINT   KK
               |
               ARC     TO      JJ
               |       |
               |       POINT   II
               |       |
               |       ARC     TO      HH
               |       |       GG
               |       |
               |       POINT   FF
               |       EE
               |
               LINE    DD
               |
               POINT   CC
               BB
```

Fig. 7 Example of Parsed Tree of Graphical Language in AED Jr.

and graphical mechanizations of language, and we expect many further developments to arise from the consideration of the two in the same framework. That the two forms of language are just different representations of the same thing may easily be seen by observing that as is shown in the May 6th demonstration the sequence "symbol word symbol word ..." is indistinguishable from the sequence "pen-position button pen-position ...". Thus for Computer-Aided Design there can in fact be a single language facility which can be molded to take whatever form is suitable to the design process itself.

CONCLUSION

The compiler, language, and system-building activities of the Computer-Aided Design part of Project MAC are continuing at a rapid pace. Since March 1, 1964, experienced system programmers from six companies (North American, Lockheed-Georgia, Grumman Aircraft, Sandia Corporation, United Aircraft, and Boeing Aircraft) have joined the Project for a year to participate in the construction of the AED-1 System, using AED-0. An additional programmer from IBM is expected in a few weeks. AED-1 will be a very efficient compiler for further extensions to the AED language for plex programming, and is being designed (using extensions to the merge process described above) to incorporate an "Algorithmic Second Pass" which will enable the system to generate very efficient machine code for essentially arbitrary computers. It is hoped that the first version of AED-1 will be operational within a year.

In parallel with the AED-1 Project, further developments of the graphical language, and extensions to the AED Jr. type of system operations will continue. Thesis activities are planned in the areas of algebraic formula manipulation, and generalized symbolic computation, as well as in many areas of design itself, both mechanical, electrical and in naval architecture.

It is hoped that this brief presentation of some of the fundamental concepts underlying the activities of the Computer-Aided Design work will engender confidence that the grandiose-sounding goals of the Project will begin to be achieved in a substantial way in the very near future. Taken in concert with the many other successful developments taking place in other parts of Project MAC, it is clear that the computer may shortly be viewed as a highly capable helper in problem-solving, rather than merely a tool for the specialists.

REFERENCES

1. Stotz, R., "Specialized Computer Equipment for Generation and Display of Three-Dimensional Curvilinear Figures," M. I. T. Report ESL-TM-167, March, 1963. (SM Thesis in Department of Electrical Engineering, M. I. T.; also published in condensed form in the Proceedings of the 1963 Spring Joint Computer Conference in Detroit.)

2. Ross, D. T., and Rodrigues, J. E., "Theoretical Foundations for the Computer-Aided Design System," M. I. T. Report ESL-TM-170, March, 1963. (Also published in the Proceedings of the 1963 Spring Joint Computer Conference in Detroit.)

3. Coons, S. A., "An Outline of the Requirements for a Computer-Aided Design System," M. I. T. Report ESL-TM-169, March, 1963. (Also published in the Proceedings of the 1963 Spring Joint Computer Conference in Detroit.)

4. Sutherland, I. E., "Sketchpad, A Man-Machine Communication System," Lincoln Report TR-396, January, 1963. (Ph. D. Thesis in Department of Electrical Engineering, M. I. T.; also published in condensed form in the Proceedings of the 1963 Spring Joint Computer Conference in Detroit.)

5. Johnson, T. E., "Sketchpad III, Three-Dimensional Graphical Communication with a Digital Computer," M. I. T. Report ESL-R-173, May, 1963. (SM Thesis in Department of Mechanical Engineering, M. I. T.; also published in condensed form in the Proceedings of the 1963 Spring Joint Computer Conference in Detroit.)

6. Ross, D. T., and Coons, S. A., "Investigations in Computer-Aided Design for Numerically Controlled Production." Interim Technical Progress Report No. 6, M. I. T. Report ESL-IR-180. Covers period 1 September 1962 through 31 May 1963, August, 1963.

7. Feldmann, C. G., "AED-0 Programmer's Guide," MAC Memorandum MAC-M-146, April 21, 1964.

8. Feldmann, C. G., "Warnings and Restrictions in AED-0," MAC Memorandum MAC-M-154, April 21, 1964.

CS-TR Scanning Project
Document Control Form

Date : 12/11/95

Report # LCS-TR-4

Each of the following should be identified by a checkmark:
Originating Department:

☐ Artificial Intellegence Laboratory (AI)
☒ Laboratory for Computer Science (LCS)

Document Type:

☒ Technical Report (TR) ☐ Technical Memo (TM)
☐ Other:_____

Document Information

Number of pages: 30 (35-IMAGES)
Not to include DOD forms, printer intstructions, etc... original pages only.

Originals are:
☐ Single-sided or
☒ Double-sided

Intended to be printed as :
☐ Single-sided or
☒ Double-sided

Print type:
☐ Typewriter ☐ Offset Press ☐ Laser Print
☐ InkJet Printer ☒ Unknown ☐ Other:_____

Check each if included with document:
☐ DOD Form ☐ Funding Agent Form ☒ Cover Page
☐ Spine ☐ Printers Notes ☐ Photo negatives
☐ Other: _____

Page Data:

Blank Pages(by page number): FOLLOWS TITLE & ABSTRACT PAGES.

Photographs/Tonal Material (by page number):_____

Other (note description/page number):

Description :	Page Number:
IMAGE MAP: (1-30) UNNUMBERED TITLE, BLANK, ABSTRACT, BLK PAGE,	1-26
(31-35) SCANCONTROL COVER, TRGT'S (3)	

Scanning Agent Signoff:

Date Received: 12/11/95 Date Scanned: 1/19/96 Date Returned: 1/26/96

Scanning Agent Signature: Michael W. Cook

Scanning Agent Identification Target

Scanning of this document was supported in part by the **Corporation for National Research Initiatives,** using funds from the **Advanced Research Projects Agency** of the **United states Government** under Grant: **MDA972-92-J1029.**

The scanning agent for this project was the **Document Services** department of the **M.I.T Libraries.** Technical support for this project was also provided by the **M.I.T. Laboratory for Computer Sciences.**

Scanned

Date: 1/19/1996

M.I.T. Libraries
Document Services